4

How Fast Is the Simplex Method?

The subject of this chapter is the number of iterations in the simplex method. We shall also comment on the distinction between theoretically satisfactory and practically satisfactory algorithms, with a particular regard to linear programming.

TYPICAL NUMBER OF ITERATIONS

For *practical* problems of the form

$$\begin{aligned} &\text{maximize} && \sum_{j=1}^{n} c_j x_j \\ &\text{subject to} && \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad (i = 1, 2, \ldots, m) \\ & && x_j \geq 0 \qquad (j = 1, 2, \ldots, n) \end{aligned} \tag{4.1}$$

with $m < 50$ and $m + n < 200$, Dantzig (1963, p. 160) reported the number of iterations as being usually less than $3m/2$ and only rarely going to $3m$. This observation agrees with empirical findings obtained more recently for much larger problems: the

typical number of iterations increases proportionally to m (with the proportionality constant in the range suggested by Dantzig) and only very slowly with n. (It is sometimes said that, for a fixed m, the typical number of iterations is proportional to the logarithm of n.) Theoretical explanations of this phenomenon were proposed by G. B. Dantzig (1980), K.-H. Borgwardt (1982) and S. Smale (1982). It is this remarkable efficiency of the simplex method that accounts for its staggering success. At the current level of computer technology, typical problems with about 100 constraints and variables are solved in a few seconds; even problems with several thousands of constraints can be handled successfully. (To attain this level of efficiency, the simplex method has to be implemented properly, so that the time *per iteration* is reduced as much as possible. Consequently, the format of dictionaries has to be abandoned in favor of less time-consuming ways of organizing the necessary computations. We shall begin to study this matter in Chapter 7.) For problems with some particular structure amenable to specialized versions of the simplex method (such as the network simplex method of Chapter 19 or generalized upper bounding of Chapter 25), this limit can be pushed even further.

Monte Carlo simulation studies of the number of iterations were pioneered by H. W. Kuhn and R. E. Quandt (1963), who solved a number of problems (4.1) with $c_j = 1$ for all j, $b_i = 10{,}000$ for all i, and each a_{ij} selected at random from the set of positive integers between 1 and 1,000. A small part of these experiments has been reproduced, on a slightly larger scale, with the results exhibited in Table 4.1. (Each entry in the table represents the average number of iterations over 100 problems.) In each simplex iteration, the entering variable was that nonbasic variable that had the largest coefficient in the z-row of the dictionary. We shall refer to this selection rule as the *largest-coefficient* rule.

TABLE 4.1 Average Number of Iterations Required by the Largest-Coefficient Rule

m \ n	10	20	30	40	50
10	9.40	14.2	17.4	19.4	20.2
20		25.2	30.7	38.0	41.5
30			44.4	52.7	62.9
40				67.6	78.7
50					95.2

Source: D. Avis and V. Chvátal (1978).

The production management problems solved in practice are very much different from such randomly generated examples. Typically, most of their coefficients a_{ij} are zeros, the remaining nonzero coefficients occur in clusters that are very far from random, and the range of distinct numerical values of the coefficients is often very small. In spite of these differences, the Monte Carlo simulation results are in striking agreement with the empirical observations quoted above: for instance, if $n = 50$, then the average number of iterations is about $2m$.

PROBLEMS REQUIRING AN UNUSUALLY LARGE NUMBER OF ITERATIONS

From a purist point of view, it would be even more reassuring to have a proof that, for *every* problem (4.1), the simplex method would require no more than, say, $10mn$ iterations to find an optimal solution. However, there is no such proof. Worse than that, there are examples of LP problems that make the simplex method go through an enormous number of iterations. V. Klee and G. J. Minty (1972) have shown that in the process of solving the problem

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} 10^{n-j} x_j \\ \text{subject to} \quad & \left(2 \sum_{j=1}^{i-1} 10^{i-j} x_j\right) + x_i \leq 100^{i-1} \qquad (i = 1, 2, \ldots, n) \\ & x_j \geq 0 \qquad (j = 1, 2, \ldots, n) \end{aligned} \tag{4.2}$$

the simplex method goes through $2^n - 1$ iterations. (A proof is outlined in problems 4.2 and 4.3.) This number is quite frightening. For example, at the rate of 100 iterations per second (a reasonably generous estimate), problem (4.2) with $n = 50$ would take more than 300,000 years to solve! (The empirical and simulation results just quoted do *not* contradict this result. They simply suggest that problems requiring large numbers of iterations must be rare. For this reason, the Klee–Minty examples (4.2) and other similar examples are sometimes referred to as "pathological.")

As our starting point for further discussion, we choose the Klee–Minty problem with $n = 3$,

$$\begin{aligned} \text{maximize} \quad & 100x_1 + 10x_2 + x_3 \\ \text{subject to} \quad & x_1 \leq 1 \\ & 20x_1 + x_2 \leq 100 \\ & 200x_1 + 20x_2 + x_3 \leq 10{,}000 \\ & x_1, x_2, x_3 \geq 0. \end{aligned} \tag{4.3}$$

Using the largest-coefficient rule, we construct the following sequence of dictionaries. The initial dictionary:

$$\begin{aligned} x_4 &= 1 - x_1 \\ x_5 &= 100 - 20x_1 - x_2 \\ x_6 &= 10{,}000 - 200x_1 - 20x_2 - x_3 \\ \hline z &= 100x_1 + 10x_2 + x_3. \end{aligned}$$

After the first iteration:

$$\begin{aligned} x_1 &= 1 - x_4 \\ x_5 &= 80 + 20x_4 - x_2 \\ x_6 &= 9{,}800 + 200x_4 - 20x_2 - x_3 \\ \hline z &= 100 - 100x_4 + 10x_2 + x_3. \end{aligned}$$

After the second iteration:

$$\begin{aligned} x_1 &= 1 - x_4 \\ x_2 &= 80 + 20x_4 - x_5 \\ x_6 &= 8{,}200 - 200x_4 + 20x_5 - x_3 \\ \hline z &= 900 + 100x_4 - 10x_5 + x_3. \end{aligned}$$

After the third iteration:

$$\begin{aligned} x_4 &= 1 - x_1 \\ x_2 &= 100 - 20x_1 - x_5 \\ x_6 &= 8{,}000 + 200x_1 + 20x_5 - x_3 \\ \hline z &= 1{,}000 - 100x_1 - 10x_5 + x_3. \end{aligned}$$

After the fourth iteration:

$$\begin{aligned} x_4 &= 1 - x_1 \\ x_2 &= 100 - 20x_1 - x_5 \\ x_3 &= 8{,}000 + 200x_1 + 20x_5 - x_6 \\ \hline z &= 9{,}000 + 100x_1 + 10x_5 - x_6. \end{aligned}$$

After the fifth iteration:

$$\begin{aligned} x_1 &= 1 - x_4 \\ x_2 &= 80 + 20x_4 - x_5 \\ x_3 &= 8{,}200 - 200x_4 + 20x_5 - x_6 \\ \hline z &= 9{,}100 - 100x_4 + 10x_5 - x_6. \end{aligned}$$

After the sixth iteration:

ionaries.

$$
\begin{aligned}
x_1 &= 1 - x_4 \\
x_5 &= 80 + 20x_4 - x_2 \\
x_3 &= 9{,}800 + 200x_4 - 20x_2 - x_6 \\
\hline
z &= 9{,}900 + 100x_4 - 10x_2 - x_6.
\end{aligned}
$$

After the seventh iteration:

$$
\begin{aligned}
x_4 &= 1 - x_1 \\
x_5 &= 100 - 20x_1 - x_2 \\
x_3 &= 10{,}000 - 200x_1 - 20x_2 - x_6 \\
\hline
z &= 10{,}000 - 100x_1 - 10x_2 - x_6.
\end{aligned}
$$

In the first iteration, we were led to an unfortunate choice of the entering variable: had we made x_3 rather than x_1 enter the basis, we would have pivoted directly to the final dictionary. In view of this blunder, it is natural to question the expediency of the largest-coefficient rule: perhaps the simplex method would *always* go through only a small number of iterations if it were directed by some other rule. In fact, the largest-coefficient rule is not quite natural. More specifically, it ranks the potential candidates for entering the basis according to their coefficients in the last row of the dictionary: variables with larger coefficients appear to be more promising. But appearances are misleading and the ranking order is easily upset by changes in the scale on which each candidate is measured. For instance, the substitution

$$\bar{x}_1 = x_1, \quad \bar{x}_2 = 0.01x_2, \quad \bar{x}_3 = 0.0001x_3$$

converts the Klee–Minty problem (4.3) into the form

$$
\begin{aligned}
\text{maximize} \quad & 100\bar{x}_1 + 1{,}000\bar{x}_2 + 10{,}000\bar{x}_3 \\
\text{subject to} \quad & \bar{x}_1 \leq 1 \\
& 20\bar{x}_1 + 100\bar{x}_2 \leq 100 \\
& 200\bar{x}_1 + 2{,}000\bar{x}_2 + 10{,}000\bar{x}_3 \leq 10{,}000 \\
& \bar{x}_1, \bar{x}_2, \bar{x}_3 \geq 0.
\end{aligned}
$$

In the first dictionary associated with this new version of (4.3), the nonbasic variable $\bar{x}_3$ appears most attractive, and so the simplex method reaches the optimal solution in only one iteration.

ALTERNATIVE PIVOTING RULES

Thus we are led to ranking the candidates x_j for entering the basis according to criteria that are independent of changes of scale. One criterion of this kind is the increase in the objective function obtained when x_j actually enters the basis. The

resulting rule (always choose that candidate whose entrance into the basis brings about the largest increase in the objective function) is referred to as the *largest-increase* rule. On the Klee–Minty examples (4.2), the largest-increase rule leads the simplex method to the optimal solution in only one iteration, as opposed to the $2^n - 1$ iterations required by the previously used largest-coefficient rule. However, the new rule does not always lead to a small number of iterations: R. G. Jeroslow (1973) constructed LP problems that are to the largest-increase rule what the Klee–Minty problems are to the largest-coefficient rule. (More precisely, the number of iterations required by the largest-increase rule grows exponentially with m and n.) Again, these examples exploit the myopia inherent in the simplex method. It is conceivable that every easily implemented rule for choosing the entering variable can be tricked in a similar way into requiring very large numbers of iterations.

Which of the two rules is better? On problems arising from applications, the number of iterations required by the largest increase is *usually* smaller than the number of iterations required by the largest coefficient. Simulation experiments lead to a similar outcome (see Table 4.2).

Table 4.2 Average Numbers of Iterations Required by the Largest-Increase Rule

m \ n	10	20	30	40	50
10	7.02	9.17	10.8	12.1	12.6
20		16.2	20.2	24.2	27.3
30			28.7	34.5	39.4
40				43.3	39.9
50					58.9

Source: D. Avis and V. Chvátal (1978).

Nevertheless, as the largest-coefficient rule takes less time to execute than the largest increase, it is the former that usually wins in terms of total computing time. More generally, the number of iterations is a poor criterion for assessing the efficiency of a rule for choosing the entering variable. It is the total computing time that counts, and rules that tend to reduce the number of iterations often take too much time to execute. In this light, even the largest-coefficient rule is found too time-consuming and therefore rarely, if ever, used in practice. The choice of entering variables in efficient implementations of the simplex method is influenced by the logistics of handling large problems on a computer; this matter will be studied in Chapter 7.

A systematic rule that always leads to an unambiguous choice of the entering variable, and to an unambiguous choice of the leaving variable in case of a tie, is called a *pivoting rule*. The largest-coefficient rule and the largest-increase rule, amended by unambiguous instructions for tie-breaking, are two examples of pivoting rules; the smallest-subscript rule of Chapter 3 is another.

EFFICIENCY OF ALGORITHMS IN THEORY AND PRACTICE

As noted in Chapter 1, the theoretical and the practical criteria for judging the efficiency of algorithms are radically different. From the theoretical point of view, an algorithm is satisfactory if its running time increases only slowly with the size of the problem. This is a vague definition as it stands; we are going to make it precise.

Let us consider a fixed class of problems (such as linear programming problems in the standard form) and a fixed algorithm (such as the simplex method) for solving problems in this class. A fair interpretation of the "size of the problem" is the time required to transmit the data. To put it differently, the size of a problem is the number of times you have to hit the keyboard of your typewriter in order to write down the data. For instance, the size of the Klee–Minty problems (4.2) is roughly $n^3/3$ when n gets very large (each of the $i - j + 1$ digits in each coefficient $2 \cdot 10^{i-j}$ has to be written down). A fair interpretation of "running time" is the total number of elementary steps (such as adding up, multiplying, or comparing two one-digit numbers; executing a "go to" instruction in a computer program; and so on) that have to be executed. (Thus it is implicitly assumed that each elementary step requires one unit of time.) Now for each s, there may be many (but only finitely many) different problems of size s in our class, and our algorithm may require different amounts of time $t_1, t_2, \ldots, t_M$ for different problems $P_1, P_2, \ldots, P_M$ of this size. Only the largest of these numbers t_i matters in the theoretical context. Of course, this largest t_i depends on s; we shall denote it by $t(s)$. Thus, our algorithm solves every problem of size s within $t(s)$ units of time and actually uses up these $t(s)$ units of time in the worst case. Finally, the algorithm is considered satisfactory if $t(s)$ grows only slowly with s. More precisely, the algorithm is satisfactory if there is a polynomial p such that $t(s) \leq p(s)$ for all s.

This definition, proposed by J. Edmonds (1965), is one of the most fruitful and stimulating concepts in theoretical computer science. [Those wishing for more information on this subject are referred to Garey and Johnson (1979).] Nevertheless, even though this concept does reflect to some extent the reasons why practitioners are satisfied by some algorithms and unsatisfied by others, it fails to capture these reasons fully. Two of the features that make it unrealistic from a practical point of view are:

(i) The worst-case criterion.

(ii) The asymptotic point of view.

The inadequacy of the worst-case criterion is demonstrated most dramatically on the case of the simplex method itself: even eminently useful algorithms may be labeled unsatisfactory on the basis of a few isolated examples of a kind that might never come up in practice. The average running time ($\sum t_i/M$) might provide a more realistic criterion than the worst running time ($\max t_i$); unfortunately, a rigorous analysis of the average performance is often much more

difficult than an analysis of the worst performance. The inadequacy of the second feature may manifest itself even when the running time depends only on the size of the problem, so that the average performance and the worst performance coincide. The point is that the actual values $t(s)$, with s restricted to a finite range, do not matter at all; the only thing that counts is the rate of growth of $t(s)$ as s increases beyond every bound. Thus, a hypothetical algorithm with a running time $t = 10^{s/1,000,000,000}$ (rounded up to the nearest integer) would be found theoretically unsatisfactory even though $t(s) \leq 10$ whenever $s \leq 10^9$; on the other hand, an algorithm with a running time $t(s) = 10^{1,000}s$ would be found theoretically satisfactory even though $t(s) \geq 10^{1,000}$ for all s. Theorists judge algorithms by their worst performance on problems of sizes outside the range of practical interest, whereas practitioners judge algorithms by their typical performance on problems whose sizes are limited to a finite range. (In all fairness, it should be admitted that the theoretical definition is not all that bad. As it turns out, the polynomials bounding the running time of theoretically satisfactory algorithms often assume reasonably small values for reasonably small values of s and, on the other hand, even the typical running time of theoretically unsatisfactory algorithms will often get out of hand already for small values of s.)

For many years, while practitioners were trying to reduce the typical running time of the simplex method by yet another 10% or 20%, theorists were trying to answer a fundamental question: Is there a theoretically satisfactory algorithm for solving linear programming problems? Eventually, L. G. Khachian (1979) provided an affirmative answer by presenting such an algorithm. This "ellipsoid method" is surprisingly simple and elegant; we shall describe its details in the appendix. Will this beautiful gem of pure mathematics ever become a serious challenger of the simplex method's supremacy in solving practical LP problems? That remains to be seen; □ at the time of this writing, it seems very likely that the answer is no.

PROBLEMS

△ **4.1** Compare the performance of the three pivoting rules discussed in this chapter on the following examples:

a. maximize $4x_1 + 5x_2$
subject to $2x_1 + x_2 \leq 9$
$x_1 \leq 4$
$x_2 \leq 3$
$x_1, x_2 \geq 0$

b. maximize $2x_1 + x_2$
subject to $3x_1 + x_2 \leq 3$
$x_1, x_2 \geq 0$

c. maximize $3x_1 + 5x_2$
subject to $x_1 + 2x_2 \leq 5$
$x_1 \leq 3$
$x_2 \leq 2$
$x_1, x_2 \geq 0.$

4.2 In the Klee–Minty
by $x_{n+1}, x_{n+2}, \ldots,$
variables x_i, s_i is bas

4.3 Use the result of pr
with the largest coeff
properties:

(i) After $2^{n-1} - 1$

$z = 10($ 100

(ii) After 2^{n-1} itera

$z = 90 \cdot 100$

(iii) After $2^n - 1$ it

$z = 100^{n-1}$

(iv) After each itera

4.2 In the Klee–Minty problem (4.2), denote the slack variables by $s_1, s_2, \ldots, s_n$ rather than by $x_{n+1}, x_{n+2}, \ldots, x_{2n}$. Prove that in every feasible dictionary, precisely one of the two variables x_i, s_i is basic.

4.3 Use the result of problem 4.2 and induction on n to prove that, when the simplex method with the largest coefficient rule is applied to (4.2), the resulting dictionaries have the following properties:

(i) After $2^{n-1} - 1$ iterations, the last row reads

$$z = 10\left(100^{n-2} - \sum_{j=1}^{n-2} 10^{n-1-j}x_j - s_{n-1}\right) + x_n.$$

(ii) After 2^{n-1} iterations, the last row reads

$$z = 90 \cdot 100^{n-2} + 10\left(\sum_{j=1}^{n-2} 10^{n-1-j}x_j + s_{n-1}\right) - s_n.$$

(iii) After $2^n - 1$ iterations, the last row reads

$$z = 100^{n-1} - \sum_{j=1}^{n-1} 10^{n-j}x_j - s_n.$$

(iv) After each iteration, all the coefficients in the last row are integers.